Minimisation de la somme des retards pour les problèmes d'ordonnancement à une machine

Chengbin Chu, Marie-Claude Portmann

INRIA

UNITE DE RECHERCHE
INRIA-LORRAINE

Rapports de Recherche

N° 1023

Programme 5

MINIMISATION DE LA SOMME DES RETARDS POUR LES PROBLEMES D'ORDONNANCEMENT A UNE MACHINE

Chengbin CHU
Marie-Claude PORTMANN

Avril 1989

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tel. (1) 39 63 55 11

Minimisation de la somme des retards pour les problèmes d'ordonnancement à une machine

Minimizing Total Tardiness for the One-Machine Scheduling Problem

CHU Chengbin (*) & PORTMANN Marie-Claude (**)

Résumé:

Dans cet article, nous démontrons un théorème qui présente une condition suffisante d'optimalité locale pour le problème d'ordonnancement du type $n/1/r_i/\Sigma T_i$. Cette condition nous permet de définir un nouveau sous-ensemble dominant de solutions pour ce problème qui est NP-difficile. Nous utilisons les résultats obtenus pour construire un algorithme approché polynômial et donnons une majoration de l'erreur commise par cet algorithme dans le pire des cas.

Mots clés:

Ordonnancement - Séquencement - Arrivées échelonnées - Minimisation somme des retards - Propriétés de dominance - Méthodes approchées - Analyse dans le pire des cas

Abstract:

In this paper, we provide a theorem giving a sufficient condition for local optimality for the $n/1/r_i/\Sigma T_i$-type scheduling problem. This condition permits us to define a new dominant subset of schedules for this NP-hard problem. We propose a new efficient heuristic based on the theorem and provide an upper bound for the error made by the heuristic in the worst case.

Key words:

Scheduling - Sequencing - Release dates - Mean tardiness minimization - Dominance properties - Approximate methods - Worst case analysis

---

(*) Projet SAGEP, INRIA-LORRAINE, Campus Scientifique B.P. 239, 54506 VANDŒUVRE CEDEX
(**) Projet SAGEP, INRIA-LORRAINE, Campus Scientifique B.P. 239, 54506 VANDŒUVRE CEDEX
et Ecole des Mines de NANCY, Parc de Saurupt, 54042 NANCY CEDEX

1. Introduction

Dans cet article, nous considérons le problème d'ordonnancement déterministe du type $n/1/r_i/\Sigma T_i$, où n est le nombre de tâches, r_i est la date d'arrivée de la tâche i, T_i est le retard correspondant à la tâche i pour une solution donnée. Le problème consiste à trouver un ensemble de dates de début d'exécution t_i (ou de fin d'exécution c_i) pour chaque tâche i afin de minimiser la somme des retards.
Le critère s'écrit donc:

$$\text{Min}(\sum_{i=1}^{n}\max(t_i+p_i-d_i,0))=\text{Min}(\sum_{i=1}^{n}\max(c_i-d_i,0))$$

où p_i est le temps nécessaire pour réaliser la tâche i et d_i le délai non impératif de la tâche i.

Les problèmes d'ordonnancement avec minimisation de la somme des retards sont combinatoires. Même dans le cas d'une seule machine, le problème est NP-difficile quand les dates d'arrivée des tâches ne sont pas identiques ([4]).

Dans ce paragraphe, nous rappelons les résultats connus pour le cas où les dates d'arrivée sont identiques. Il existe des algorithmes polynômiaux optimaux pour des cas très particuliers, par exemple la règle EDD (Earliest Due Date) donne une solution optimale quand toutes les tâches ont la même durée ([1], [2] et [3]), mais dans le cas général, la complexité du problème reste un problème ouvert ([4]). On trouve plusieurs algorithmes exacts ou approchés dans la littérature ([1]): l'algorithme de Wilkerson-Irwin, construit à partir de conditions d'optimalité locale, est polynômial mais, comme les théorèmes de dominance correspondants ne couvrent pas tous les cas possibles, il ne donne pas toujours la solution optimale. L'algorithme hybride de Srinivasan ([1], [6]) conduit à une solution optimale. Il consiste à ordonnancer les tâches en deux phases: dans la première, on obtient un ordre à respecter pour une partie des tâches de la séquence en utilisant des propriétés de dominance; dans la deuxième phase on complète la séquence en utilisant la programmation dynamique. Cette deuxième phase est combinatoire.

Dans cet article, nous démontrons une condition suffisante d'optimalité locale pour le problème d'ordonnancement du type $n/1/r_i/\Sigma T_i$ et proposons une nouvelle heuristique efficace construite à partir de cette condition.

La section 2 présente le théorème. La section 3 propose l'heuristique. La section 4 discute de la complexité de l'heuristique et propose une majoration de l'erreur pouvant être commise dans le pire des cas.

2. Condition suffisante d'optimalité locale

Nous nous intéressons ici à deux tâches consécutives i et j dans un ordonnancement donné et nous cherchons à quelle condition placer i avant j minimise la somme des retards de ces deux tâches (*). Le problème consiste à placer de manière optimale i et j à partir de l'instant Δ où la machine est libérée par les tâches qui précèdent.

En notant $T_{i,j}(\Delta)$ (respectivement $T_{j,i}(\Delta)$) la somme des retards des tâches i et j en plaçant, au plus tôt à partir de l'instant Δ, i avant j (resp. j avant i), on cherche à quelle condition l'on a $T_{i,j}(\Delta) \leq T_{j,i}(\Delta)$.

Nous allons montrer que la condition cherchée est obtenue grâce à une nouvelle règle de priorité de chaque tâche face aux délais que nous appelons PRIOR.

Définition 1:

On définit la fonction *"priorité face aux délais"* PRIOR(i,Δ) (**) pour la tâche i commençant au plus tôt à l'instant Δ par

$$PRIOR(i,\Delta)=\max(\Delta,r_i) + \max[\max(\Delta,r_i)+p_i, d_i]$$

Théorème 1:

Nous considérons le problème partiel d'ordonnancement de deux tâches i et j sur une machine disponible à partir de l'instant Δ.

Si nous avons $PRIOR(i,\Delta) \leq PRIOR(j,\Delta)$ alors placer i avant j minimise la somme des retards des deux tâches i et j.

C'est-à-dire:

$$PRIOR(i,\Delta) \leq PRIOR(j,\Delta) \Rightarrow T_{i,j}(\Delta) \leq T_{j,i}(\Delta)$$

Démonstration:

En notant $R_k=\max(\Delta,r_k)$ le début d'exécution au plus tôt de la tâche k, alors la définition de PRIOR(k,Δ) devient:

$$PRIOR(k,\Delta)=R_k+\max(R_k+p_k,d_k)$$

et celle de $T_{i,j}(\Delta)$ (que nous noterons pour simplifier T_{ij} lorsqu'il n'y a pas d'ambiguïté) devient:

$$T_{ij}=\max(R_i+p_i-d_i,0)+\max[\max(R_i+p_i,R_j)+p_j-d_j,0]$$

(*) Dans cette recherche, on ne prend pas en compte les conséquences de l'échange de i et de j sur les retards éventuels des tâches qui les suivent.

(**) Une étude analytique de la fonction PRIOR est fournie en annexe.

que l'on peut réécrire:

$$T_{ij}=\max(R_i+p_i,d_i)+\max[\max(R_i+p_i+p_j,R_j+p_j),d_j]-(d_i+d_j)$$

ce qui donne en utilisant l'associativité de l'opération "max":

$$T_{ij}=\max(R_i+p_i,d_i)+\max[R_i+p_i+p_j,\max(R_j+p_j,d_j)]-(d_i+d_j)$$

puis: $T_{ij}=\max[R_i+\max(R_i+p_i,d_i)+p_i+p_j,\max(R_i+p_i,d_i)+\max(R_j+p_j,d_j)]-(d_i+d_j)$

$$=\max[PRIOR(i,\Delta)+p_i+p_j,\max(R_i+p_i,d_i)+\max(R_j+p_j,d_j)]-(d_i+d_j)$$

De manière symétrique, on obtiendrait:

$$T_{ji}=\max[PRIOR(j,\Delta)+p_i+p_j,\max(R_i+p_i,d_i)+\max(R_j+p_j,d_j)]-(d_i+d_j)$$

En posant:

$P_{ij}=p_i+p_j$,

$Q_{ij}=\max(R_i+p_i,d_i)+\max(R_j+p_j,d_j)$

et $D_{ij}=d_i+d_j$

ces trois expressions sont symétriques en i et en j et on a:

$$T_{ij}=\max(PRIOR(i,\Delta)+P_{ij},Q_{ij})-D_{ij} \quad (1)$$

et $$T_{ji}=\max(PRIOR(j,\Delta)+P_{ij},Q_{ij})-D_{ij} \quad (2)$$

d'où $PRIOR(i,\Delta)\leq PRIOR(j,\Delta) \Rightarrow T_{ij}\leq T_{ji}$

Q.E.D.

La condition que nous venons de démontrer est une condition suffisante d'optimalité locale. Il est possible de montrer que cette condition est également nécessaire lorsque des conditions que nous allons préciser sont réunies.

Pour démontrer cette réciproque, nous allons introduire la notion de "délai corrigé" et montrer que l'ensemble des solutions optimales pour les délais corrigés est analogue à l'ensemble des solutions optimales pour les délais originaux, ce qui nous conduit à affirmer que toutes les méthodes d'ordonnancement qui cherchent à minimiser la somme des retards pourraient n'utiliser que les délais corrigés.

Définition 2:

On définit la notion de *"délai corrigé"* d_i^c de la tâche i par

$$d_i^c=\max(r_i+p_i,d_i) \quad (3)$$

Cette quantité est le maximum entre le délai et l'instant de fin de fabrication au plus tôt.

Lemme 1:

Si l'on considère un problème P de minimisation de la somme des retards et le problème P' obtenu en remplaçant les délais d_i par les délais corrigés d_i^c, alors les problèmes P et P' sont équivalents.

Démonstration:

On note T'_i le retard calculé avec d_i^c de la tâche i, et T' la somme des retards calculés avec les "délais corrigés". On démontre ce lemme en distinguant les deux seuls cas possibles selon que le retard de toute tâche i est inévitable ou non.

1) cas où $r_i+p_i \geq d_i$

$(3) \Rightarrow d_i^c = r_i+p_i$ (4)

$t_i \geq r_i$ et $(4) \Rightarrow t_i+p_i \geq r_i+p_i = d_i^c \geq d_i$

D'où $T_i-T'_i = \max(t_i+p_i-d_i,0)-\max(t_i+p_i-d_i^c,0) = t_i+p_i-d_i-(t_i+p_i-d_i^c) = d_i^c-d_i$

2) cas où $r_i+p_i < d_i$

$(3) \Rightarrow d_i^c = d_i$

D'où $T_i-T'_i = \max(t_i+p_i-d_i,0)-\max(t_i+p_i-d_i^c,0) = 0 = d_i^c-d_i$

Dans tous les cas, on a:

$T_i-T'_i = d_i^c-d_i$

$$\Rightarrow T-T' = \sum_{i=1}^{n} T_i - \sum_{i=1}^{n} T'_i = \sum_{i=1}^{n} (T_i-T'_i) = \sum_{i=1}^{n} (d_i^c-d_i) = \text{CONSTANTE}$$

Q.E.D.

Théorème 2:

Nous considérons le problème partiel d'ordonnancement de deux tâches i et j sur une machine disponible à partir de l'instant Δ.

Si nous travaillons avec des "délais corrigés" et si nous avons $T_{j,i}(\Delta) > 0$ alors:

$T_{i,j}(\Delta) \leq T_{j,i}(\Delta) \Rightarrow \text{PRIOR}(i,\Delta) \leq \text{PRIOR}(j,\Delta)$

Démonstration:

Comme nous travaillons sur des délais corrigés nous avons:

$\forall k,\ d_k^c \geq r_k+p_k$ (5)

Et nous allons montrer que:

$\text{PRIOR}(i,\Delta) > \text{PRIOR}(j,\Delta) \Rightarrow T_{i,j}(\Delta) > T_{j,i}(\Delta)$ ou $T_{i,j}(\Delta) = T_{j,i}(\Delta) = 0$

Pour cela, nous considérons à nouveau les formules (1) et (2) :

$$T_{ij}=\max(PRIOR(i,\Delta)+P_{ij},Q_{ij})-D_{ij} \quad (1)$$

et $$T_{ji}=\max(PRIOR(j,\Delta)+P_{ij},Q_{ij})-D_{ij} \quad (2)$$

et nous constatons qu'il faut considérer plusieurs cas en raison de la présence du maximum.

a) $PRIOR(i,\Delta)+P_{ij} > Q_{ij}$

Alors on a:

$PRIOR(i,\Delta) > PRIOR(j,\Delta) \Rightarrow PRIOR(i,\Delta)+P_{ij} > \max (PRIOR(j,\Delta)+P_{ij} , Q_{ij})$

$\Rightarrow \max (PRIOR(i,\Delta)+P_{ij} , Q_{ij}) > \max (PRIOR(j,\Delta)+P_{ij} , Q_{ij})$

$\Rightarrow \max (PRIOR(i,\Delta)+P_{ij} , Q_{ij}) - D_{ij} > \max (PRIOR(j,\Delta)+P_{ij} , Q_{ij}) - D_{ij}$

$\Rightarrow T_{ij} > T_{ji}$

b) $PRIOR(i,\Delta)+P_{ij} \leq Q_{ij}$

En combinant avec $PRIOR(i,\Delta) > PRIOR(j,\Delta)$ on obtient:

$$T_{ij}=T_{ji}= Q_{ij} - D_{ij}$$

et on voudrait alors montrer que $Q_{ij} = D_{ij}$.

$PRIOR(i,\Delta) > PRIOR(j,\Delta) \Rightarrow PRIOR(i,\Delta)+P_{ij} > PRIOR(j,\Delta)+P_{ij}$

et $PRIOR(i,\Delta)+P_{ij} \leq Q_{ij} \Rightarrow Q_{ij} \geq PRIOR(i,\Delta)+P_{ij} > PRIOR(j,\Delta)+P_{ij}$

En développant ces dernières égalités, elles s'écrivent:

$$\max(R_i+p_i,d_i^c)+\max(R_j+p_j,d_j^c) \geq R_i+\max(R_i+p_i,d_i^c)+P_{ij} > R_j+\max(R_j+p_j,d_j^c)+P_{ij}$$

De la première inégalité on déduit:

$$\max(R_j+p_j,d_j^c) \geq R_i+P_{ij} = R_i+ p_i +p_j \quad (6)$$

Des deux termes extrêmes on déduit:

$$\max(R_i+p_i,d_i^c) > R_j+P_{ij} = R_j+ p_i +p_j \quad (7)$$

On étudie les différents cas possibles.

b1) $R_j+p_j \geq R_i+p_i+p_j$

On a dans ce cas $R_j+p_j+p_i \geq R_i+p_i +p_j+p_i > R_i+p_i$

De ces inégalités et de (7) on déduit que:

$$d_i^c > R_i+p_i \quad (8)$$

Avec l'inéquation du cas b1) on obtient:

$R_j \geq R_i+ p_i$

$\Rightarrow r_j > \Delta$ (car sinon on aurait: $R_j = \max(r_j,\Delta) = \Delta \leq \max(r_i,\Delta) =R_i < R_i+ p_i$)

D'où $r_j = R_j$ et avec (5) on obtient:

$$d_j^c \geq R_j+p_j \quad (9)$$

De (8) et (9) en revenant à la définition de Q_{ij} et de D_{ij} on obtient:

$$Q_{ij}=\max(R_i+p_i,d_i^c)+\max(R_j+p_j,d_j^c)=d_i^c+d_j^c=D_{ij}$$

b2) $R_j+p_j < R_i+p_i+p_j$

Avec (6), on retrouve la formule (9) du cas b1):

$$d_j^c \geq R_j+p_j$$

et on étudie les deux sous-cas issus de (7):

b2.1) $R_i+p_i \geq R_j+p_i+p_j$

Ce cas est symétrique au cas b1) en échangeant les indices i et j (car (6) et (7) ainsi que les autres éléments de la démonstration sont symétriques en i et j).

b2.2) $R_i+p_i < R_j+p_i+p_j$

Avec (7), on retrouve la formule (8) du cas b1):

$$d_i^c \geq R_i+p_i$$

Et (8) et (9) nous donne alors $Q_{ij} = D_{ij}$.

Q.E.D.

Nous utilisons maintenant le théorème d'optimalité locale pour définir un nouveau sous-ensemble dominant de solutions pour le critère somme des retards.

Définition 3:

Considérons un ordonnancement actif O (*) pour un problème du type $n/1/r_i/\Sigma T_i$.

Etant données i et j deux tâches consécutives (j suivant i), on pose:

$\Delta=c_k$ (où k est la tâche précédant immédiatement i)

ou

$\Delta=-\infty$ (si k n'existe pas, c'est-à-dire que i est la première tâche de la séquence).

Si pour tout couple (i,j) de tâches consécutives on a:

$\max(r_i,\Delta)<\max(r_j,\Delta)$

ou $\text{PRIOR}(i,\Delta)\leq\text{PRIOR}(j,\Delta)$

alors on dit que *l'ordonnancement O est p-actif.*

Le théorème 1 nous permet d'obtenir le corollaire suivant:

(*) Un ordonnancement est actif si, pour avancer une tâche quelconque, on est obligé de reporter le début d'une autre.

Corollaire 1:

Dans le cas du problème à une machine, le sous-ensemble des ordonnancements p-actifs est dominant pour le critère "somme des retards".

Démonstration:

La démonstration est faite par "échange de paires consécutives".

Supposons qu'il existe une solution optimale O pour un problème donné, et que O ne soit pas p-actif. Alors il existe dans O au moins une paire de tâches consécutives qui ne vérifient aucune des deux conditions de la définition de p-activité. Soit i_1 et j_1 une telle paire de tâches, alors j_1 a une plus petite priorité face aux délais que i_1 et ou bien j_1 arrive avant i_1 ou bien les deux tâches sont disponibles à l'instant Δ. Aussi selon le théorème 1, mettre j_1 avant i_1 n'augmente ni la somme des retards des deux tâches, ni la somme des retards des tâches suivantes car la fin des deux tâches n'est pas retardée. On construit une suite finie (*) d'ordonnancements en échangeant la première paire de tâches consécutives qui ne vérifie pas la p-activité dans l'ordonnancement précédent et l'on arrête lorsque toutes les paires consécutives vérifient la p-activité. On obtient un ordonnancement O' p-actif dont la somme des retards est au plus aussi grande que celle de O.

Q.E.D.

Remarque :

Si les tâches arrivent en même temps (par exemple à l'instant 0), la condition suffisante d'optimalité locale conduisant à mettre i avant j devient:

$\max(\Delta+p_i,d_i) \leq \max(\Delta+p_j,d_j)$

C'est bien la condition suffisante proposée par Smith([5]).

3. Une nouvelle heuristique

D'après le corollaire 1, on peut se contenter de considérer les ordonnancements p-actifs. Nous proposons une heuristique polynômiale qui construit une séquence p-active en appliquant le théorème 1. Elle est plus générale que l'algorithme de Wilkerson-Irwin, puisqu'elle admet des dates d'arrivée différentes.

Voici la démarche de l'algorithme:

(*) Parce que l'ensemble d'ordonnancements est fini et que l'on ne considère jamais deux fois le même.

(INITIALISATION)

1° W:=l'ensemble des indices de toutes les tâches à ordonnancer
:={1,2,3,...,n};

2° Initialiser le temps à la première arrivée de tâches:
$t:=\underset{i\in W}{MIN}(r_i)$;

(ORDONNANCEMENT PROPREMENT DIT)

3° **Tant que** $W\neq\emptyset$ **faire**

(Choix et placement définitif d'une tâche 'prioritaire')

1) Obtenir l'ensemble des tâches les plus "prioritaires":

$$W1 := \left\{ k/k\in W \text{ et } PRIOR(k,t)=\underset{j\in w}{MIN}(PRIOR(j,t)) \right\}$$

2) Choisir dans W1 une tâche l qui (placée en premier) se terminerait le plus tôt (*) :

$F(l,t)=\underset{k\in W1}{MIN}(F(k,t))$;

3) Placer l à l'instant définitif:
$t_l:=max(t,r_l)$;

4) Eliminer la tâche placée:
W:=W-{l};

(Insertion éventuelle des tâches qui pourraient être placées avant la tâche l sans la retarder)

(Initialisation de la phase d'insertion)

5) Rechercher les tâches qui peuvent être placée avant l sans la retarder:
$AV:=\{j/j\in W \text{ et } F(j,t)\leq t_l\}$;

6) Initialiser le temps pour la phase d'insertion:
t':=t;

(Insertion proprement dite)

7) Tant que $AV\neq\emptyset$ faire

7.1) Trouver le sous-ensemble de tâches de l'ensemble AV qui peuvent être exécutées au plus tôt:

$$W2:= \left\{ i/i\in AV \text{ et } max(t',r_i)=\underset{j\in AV}{MIN}max(t',r_j) \right\}$$

(*) On définit la fonction $F(i,\Delta)$, *"date de fin de la tâche i si on la place au plus tôt à partir de l'instant Δ"*, par $F(i,\Delta)=max(\Delta,r_i)+p_i$

7.2) Choisir dans W2 une tâche i parmi les plus "prioritaires":
$PRIOR(i,t') = \min_{j \in w2}(PRIOR(j,t'))$;

7.3) Placer i à l'instant définitif:
$t_i := \max(t', r_i)$;

7.4) Faire progresser le temps:
$t' := t_i + p_i$;

7.5) Eliminer la tâche placée:
AV:=AV-{i} et W:=W-{i};

7.6) Corriger AV en conséquence:
AV:=AV-{j/j∈ AV et F(j,t')>t_l};

fintantque;

8) Faire progresser le temps:
$t := t_l + p_l$;

fintantque.

Nous pouvons démontrer le théorème suivant qui caractérise une partie intéressante du fonctionnement de l'heuristique.

Théorème 3:

Aucune des tâches insérées par la phase d'insertion n'est en retard.

Démonstration:

Le théorème est évident s'il n'y a aucune tâche à insérer au cours de la phase d'insertion. Nous supposons donc qu'il en existe au moins une. La phase d'insertion de l'algorithme génère alors un sous-ordonnancement Ω comme celui illustré dans la Fig. 1.

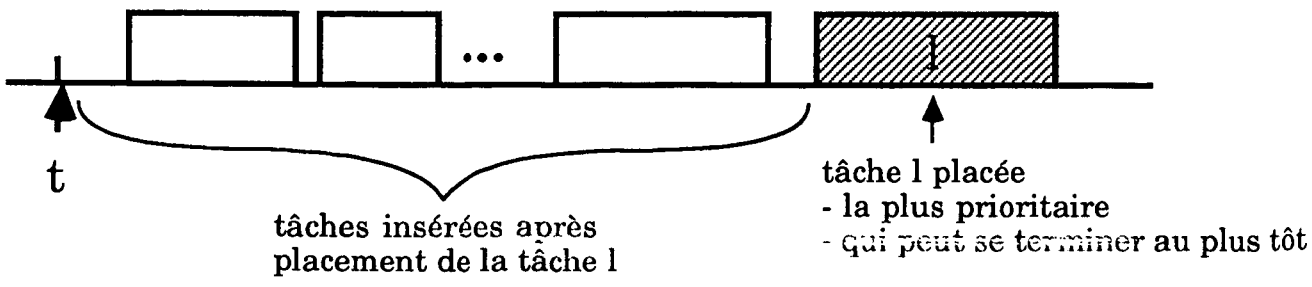

Fig. 1 Sous-ordonnancement Ω

Pour toute tâche i de Ω ($i \neq l$), i a une "priorité face aux délais" plus faible que l:

$$\max(t,r_i)+\max[\max(t,r_i)+p_i,d_i] > \max(t,r_l)+\max[\max(t,r_l)+p_l,d_l] \quad (10)$$

Par ailleurs $t_i+p_i \leq t_l$ et $\max(t,r_l)=t_l$ (*) entraînent:

$$\max(t,r_i) \leq t_i < t_l = \max(t,r_l)$$

et en combinant avec (10) on obtient:

$$d_i > \max[\max(t,r_l)+p_l,d_l] > t_i+p_i=c_i.$$

i n'est donc pas en retard.

Q.E.D.

4. Complexité de l'algorithme et majoration de l'erreur (dans le cas des durées égales)

Théorème 4:

L'algorithme décrit dans la section 3 est polynômial et sa complexité est en $O(n^2)$.

Démonstration:

L'algorithme décrit est un algorithme glouton: à chaque itération il place définitivement au moins une tâche. Le nombre d'itérations nécessaires pour aboutir à un ordonnancement complet est au plus aussi grand que le nombre n de tâches, et à chaque itération il explore au plus n fois les tâches non encore placées. La complexité de l'algorithme est de l'ordre du nombre de tâches au carré. C'est donc un algorithme polynômial en $O(n^2)$.

Q.E.D.

Théorème 5:

On considère le problème d'ordonnancement de n tâches sur une machine. Dans le cas où les durées sont égales à D (**) , si O est un ordonnancement issu de l'algorithme introduit dans la section 3 et dont la somme des retards par rapport aux délais est égale à T, et si T^* désigne la somme des retards d'une solution minimale, on a:

$$\Delta T = T - T^* \leq \frac{(2n+1)^2}{16} D$$

Démonstration:

La démonstration se déroule en quatre parties.

(*) En effet, si Ω est non vide, l doit arriver strictement après t et toute tâche i de Ω est insérable devant la tâche l.

(**) On suppose que D ne divise pas le PGCD (Plus Grand Commun Diviseur) des dates d'arrivée r_i, car dans ce cas le problème est facilement résolu par un algorithme polynômial.

Dans la première partie, on démontre des lemmes qui serviront dans la démonstration du théorème; dans la seconde partie, on analyse des caractéristiques de l'ordonnancement obtenu avec l'heuristique; dans la troisième, on cherche une fonction de majoration par rapport à des paramètres; dans la quatrième partie, on maximise cette fonction en choisissant des paramètres.

Première Partie:

Pour mieux expliquer les lemmes suivants, on introduit de nouvelles notations. On note T(P,O) la somme des retards donnée par un ordonnancement O pour un problème P, $T_i(P,O)$ le retard de la tâche i du problème P dans O et O_P^* un ordonnancement optimal pour le problème P.

Lemme 2:

Considérons un problème P et un problème P' construit à partir du problème P, en conservant toutes les données du problème P à l'exception des délais qui sont choisis supérieurs ou égaux à ceux de P:

$\forall i,\ d'_i \geq d_i$

alors on a:

$T(P',O_{P'}^*) \leq T(P,O_P^*)$

Démonstration du lemme 2:

Les problèmes P et P' ont le même ensemble de solutions réalisables. Tout ordonnancement réalisable O a une somme des retards pour P' inférieure ou égale à celle obtenue pour P ($T(P',O) \leq T(P,O)$), car si l'on augmente le délai d'une tâche, son retard ne peut que diminuer. On en déduit que: $T(P,O_P^*) \geq T(P',O_P^*) \geq T(P',O_{P'}^*)$.

Q.E.D.

Lemme 3:

Etant donné un problème P et un ordonnancement réalisable O pour P, nous construisons un problème P' en conservant les dates d'arrivée et les durées des tâches de P et en choisissant des délais pour les tâches qui vérifient:

$\forall i,\ i \in E,\ d'_i = d_i$ (11)

et $\forall i,\ i \notin E,\ d'_i \leq d_i$ (12)

où $E=\{i/c_i<d_i\}$: c'est l'ensemble des tâches strictement en avance dans O.

Alors on a:

$T(P,O)-T(P,O_P^*) \leq T(P',O)-T(P',O_{P'}^*)$

Démonstration du lemme 3:

$$T(P,O)-T(P',O)=\sum_{i=1}^{n}\max(t_i+p_i-d_i,0)-\sum_{i=1}^{n}\max(t_i+p_i-d'_i,0)$$

$$=\sum_{i\notin E}[t_i+p_i-d_i-(t_i+p_i-d'_i)]+\sum_{i\in E}0$$

$$=\sum_{i\notin E}(d'_i-d_i)+\sum_{i\in E}(d'_i-d_i)$$

$$=\sum_{i=1}^{n}(d'_i-d_i) \qquad (13)$$

Par ailleurs, on a:

$$T(P',O^*_P)\geq T(P',O^*_{P'}) \qquad (14)$$

et $$T(P',O^*_P)-T(P,O^*_P)=\sum_{i=1}^{n}\max(t^*_i+p_i-d'_i,0)-\sum_{i=1}^{n}\max(t^*_i+p_i-d_i,0)$$

$$=\sum_{i=1}^{n}[\max(t^*_i+p_i,d'_i)-d'_i]-\sum_{i=1}^{n}[\max(t^*_i+p_i,d_i)-d_i]$$

$$=\sum_{i=1}^{n}[\max(t^*_i+p_i,d'_i)-\max(t^*_i+p_i,d_i)]+\sum_{i=1}^{n}(d_i-d'_i) \qquad (15)$$

et (11) et (12) $\Rightarrow d'_i\leq d_i \Rightarrow \max(t^*_i+p_i,d'_i)\leq\max(t^*_i+p_i,d_i)$ (16)

(13), (15) et (16) $\Rightarrow T(P',O^*_P)-T(P,O^*_P)\leq\sum_{i=1}^{n}(d_i-d'_i)=T(P',O)-T(P,O)$ (17)

En combinant (14) et (17), on a:

$$T(P',O^*_{P'})-T(P,O^*_P)\leq T(P',O)-T(P,O)\Rightarrow T(P,O)-T(P,O^*_P)\leq T(P',O)-T(P',O^*_{P'})$$

Q.E.D.

Corollaire 2:

Etant donné un problème P et un ordonnancement réalisable O pour P, nous construisons un problème P' en conservant les dates d'arrivée et les durées des tâches de P, et en choisissant les délais des tâches de la manière suivante:

$\forall i,\ i\in E,\quad d'_i\geq d_i$

et $\forall i,\ i\notin E,\quad d'_i\leq d_i$

où $E=\{i/c_i<d_i\}$ l'ensemble des tâches strictement en avance dans O, alors on a:

$$T(P,O)-T(P,O^*_P)\leq T(P',O)-T(P',O^*_{P'})$$

Démonstration du corollaire 2:

Construisons un problème P" en conservant les dates d'arrivée et les durées des tâches du problème P et en choisissant les délais de la manière suivante:

$\forall i,\ i \in E,\quad d''_i = d_i$

et $\forall i,\ i \notin E,\quad d''_i = d'_i \leq d_i$

Alors, d'après le lemme 3, on a:

$$T(P,O)-T(P,O^*_P) \leq T(P'',O)-T(P'',O^*_{P''}) \qquad (18)$$

Par ailleurs, on a: $T(P'',O)=T(P',O)$ (19)

et, d'après le lemme 2, on a encore:

$$T(P',O^*_{P'}) \leq T(P'',O^*_{P''}) \qquad (20)$$

En combinant (18), (19) et (20), on a donc:

$$T(P,O)-T(P,O^*_P) \leq T(P',O)-T(P',O^*_{P'})$$

Q.E.D.

Deuxième Partie:

Le lemme 1 montre que, pour un ordonnancement quelconque, l'écart entre la somme des retards pour cet ordonnancement et la somme des retards pour la solution optimale est le même que l'on travaille avec les délais initiaux ou avec les délais corrigés. Aussi, à partir d'ici et jusqu'à la fin de l'article, nous travaillons avec des délais corrigés, mais pour alléger les notations nous les noterons simplement d_i.

Si dans O il y a au moins un intervalle de temps inutilisé de longueur supérieure ou égale à D, on peut toujours décomposer O en sous-ordonnancements indépendants les uns des autres, chaque sous-ordonnancement ne contenant que des intervalles de longueur strictement inférieure à D. En effet, O étant actif, toutes les tâches placées derrière un intervalle de longueur supérieure ou égale à D ne sont pas encore disponibles au début de cet intervalle. Le fait que la machine aurait pu être libérée plus tôt grâce à un ordonnancement différent des tâches qui précèdent n'a donc aucune influence sur les décisions ultérieures.

On ne s'intéresse donc qu'au cas où O ne laisse la machine inutilisée que pendant des intervalles de temps de longueur strictement inférieure à D, sachant que lorsqu'il y a plusieurs sous-ordonnancements indépendants, l'erreur totale est la somme des erreurs commises pour chacun d'entre eux.

On simplifie la description en choisissant l'origine de l'axe du temps de telle sorte que:

$$\min_{i=1}^{n}(r_i)=0$$

et on suppose que les tâches sont renumérotées dans l'ordre de leur rang dans O.

O est constitué de m ($m \geq 1$) sous-séquences contiguës, telles que devant chaque sous-séquence k il y a un intervalle de temps inutilisé de longueur Δ_k avec

$$\begin{cases} 0<\Delta_k<D & \forall k>1 \\ 0\leq\Delta_k<D & \text{pour } k=1 \end{cases}$$

aussi, on a:

$$t_i=(i-1)D+\sum_{k=1}^{\sigma_i}\Delta_k$$

où σ_i est le numéro de la sous-séquence à laquelle appartient la tâche i.

Considérons maintenant les caractéristiques de l'ordonnancement construit par notre heuristique.

On décide de noter δ l'ensemble des tâches placées en raison de leur plus grande priorité et δ' l'ensemble des tâches insérées lors des phases d'insertion (voir Fig.2).

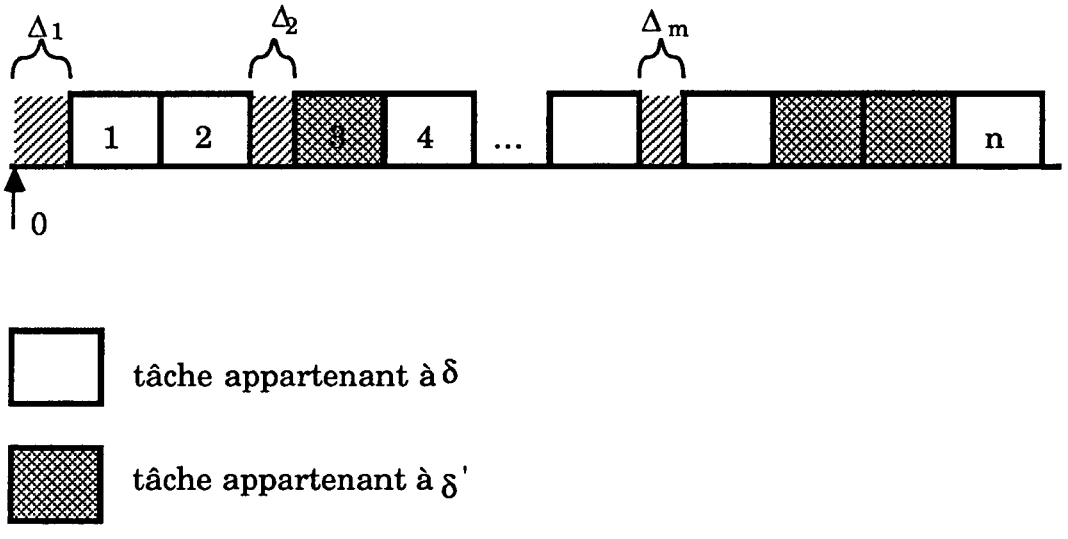

Fig.2 Ordonnancement obtenu avec l'heuristique

Nous démontrons à présent une formule qui signifie que dans l'ordonnancement fourni par l'heuristique: si une tâche j suit une tâche i qui est strictement en avance, alors j ne serait pas en retard en terminant à l'instant de fin de la tâche i. Pour cela, on considère les cas où la tâche i est dans δ puis dans δ':

$\forall i,j$, $i \in \delta$ et $j>i$, on a $PRIOR(i,t_i) \leq PRIOR(j,t_i)$, ce qui implique selon les deux cas possibles a) et b):

$$\begin{cases} a) & t_i \geq r_j \Rightarrow \max(c_i,d_j) \geq \max(c_i,d_i) \\ b) & t_i < r_j \Rightarrow d_j > c_i \end{cases}$$

et $\forall i,j$, $i \in \delta'$ et $j>i$, la règle d'insertion implique selon les deux cas possibles a) et b):

$$\begin{cases} a) & t_i \geq r_j \Rightarrow d_j \geq c_i \\ b) & t_i < r_j \Rightarrow d_j > c_i \end{cases}$$

on a donc:

$$\forall i,j, \quad j>i, \quad \text{si } d_i>c_i \text{ alors } d_j \geq c_i \tag{21}$$

Nous cherchons maintenant une formule minorant le délai de toute tâche numérotée i.

Soit h_k la première tâche de la sous-séquence k, alors:

$$\begin{cases} \forall i,\ \sigma_{h_{\sigma_i}}=\sigma_i,\ h_{\sigma_i+1}>i \geq h_{\sigma_i} \geq \sigma_i \\ \forall i,\ 1 \leq i \leq n-1,\ \sigma_i \leq \sigma_{i+1} \leq \sigma_i+1 \end{cases} \tag{22}$$

Pour tout i, on a encore (car on travaille sur les délais corrigés: $d_i \geq r_i+D$)

$$\begin{cases} \text{si } i=h_{\sigma_i},\ r_i=t_i=(i-1)D+\sum_{k=1}^{\sigma_i}\Delta_k \Rightarrow d_i \geq r_i+D=h_{\sigma_i}D+\sum_{k=1}^{\sigma_i}\Delta_k \\ \text{si } i>h_{\sigma_i}, \begin{cases} \text{si } r_i \geq r_{h_{\sigma_i}} \Rightarrow d_i \geq r_i+D \geq r_{h_{\sigma_i}}+D=h_{\sigma_i}D+\sum_{k=1}^{\sigma_i}\Delta_k \\ \text{si } r_i<r_{h_{\sigma_i}},\ d_i>d_{h_{\sigma_i}} \geq h_{\sigma_i}D+\sum_{k=1}^{\sigma_i}\Delta_k \quad (*) \end{cases} \end{cases}$$

c'est-à-dire une formule de minoration du délai de la $i^{ème}$ tâche:

$$\forall i, \quad d_i \geq h_{\sigma_i}D+\sum_{k=1}^{\sigma_i}\Delta_k \tag{23}$$

(*) Dans ce cas, on a:

$PRIOR(h_{\sigma_i},t_{h_{\sigma_i}}-\Delta_{\sigma_i}) < PRIOR(i,t_{h_{\sigma_i}}-\Delta_{\sigma_i}) \Rightarrow r_{h_{\sigma_i}}+d_{h_{\sigma_i}} < \max(r_i,t_{h_{\sigma_i}}-\Delta_{\sigma_i})+\max[\max(r_i,t_{h_{\sigma_i}}-\Delta_{\sigma_i})+D,d_i]$

et par ailleurs, $\max(r_i,t_{h_{\sigma_i}}-\Delta_{\sigma_i}) < r_{h_{\sigma_i}}$

Il y a obligatoirement: $d_i > d_{h_{\sigma_i}}$

Troisième Partie:

Si B désigne la somme de tous les intervalles, c'est-à-dire: $B=\sum_{k=1}^{m}\Delta_k$, et si q est le quotient de la division entière de B par D, nous notons λ la valeur de q augmentée de 1, on a:

$(\lambda-1)D\leq B<\lambda D$

où λ est un entier strictement positif. Il est évident que $\lambda\leq m$, car sinon, il y aurait au moins un intervalle de temps de longueur supérieure ou égale à D.

Nous cherchons une majoration de l'erreur commise en fonction de λ.

Soit P' le problème obtenu à partir du problème P en conservant les dates d'arrivée et les durées des tâches mais en choisissant les délais de la manière suivante:

$$\forall i,\quad d'_i=d_i+\min(i,\lambda)D-\sum_{k=1}^{\sigma_i}\Delta_k \qquad (d'_i\geq d_i) \tag{24}$$

alors le lemme 2 nous permet de déduire que:

$$T(P',O^*_{P'})\leq T^* \tag{25}$$

On construit un nouvel ordonnancement O' (Fig.3.2) en décalant vers la droite chaque tâche i de O de la durée $\min(i,\lambda)D-\sum_{k=1}^{\sigma_i}\Delta_k$. Ceci conduit à mettre λ intervalles de longueur D devant les λ premières tâches, les tâches restantes étant alors toutes consécutives.

On a pour tout i:

$$\begin{aligned}T_i(P',O')&=\max(c'_i-d'_i,0)=\max[(\min(i,\lambda)+i)D-d'_i,0]\\&=\max\{[\min(i,\lambda)+i]D-[\min(i,\lambda)D-\sum_{k=1}^{\sigma_i}\Delta_k+d_i],0\}\\&=\max(iD+\sum_{k=1}^{\sigma_i}\Delta_k-d_i,0)\\&=T_i\end{aligned}$$

et on a donc:

$$T(P',O')=T \tag{26}$$

avec (25) et (26) on a donc:

$$T-T^*\leq T(P',O')-T(P',O^*_{P'}) \tag{27}$$

Il suffit maintenant de majorer $T(P',O')-T(P',O^*_{P'})$.

Soit $E'=\{i/c'_i<d'_i\}\cup\{0\}=\{0,e_1,e_2,\ldots,e_\omega\}$, où e_τ est le numéro de la τ-ième tâche strictement en avance par rapport à son délai dans O' pour le problème P'.

Soit $\tau_i=\text{card}\{l/0<l\leq i \text{ et } l\in E'\}$ le nombre de tâches strictement en avance situées avant la tâche i+1 dans O'. Alors e_{τ_i} est la tâche strictement en avance la plus proche de i et on a:

$$\begin{cases}\forall i,\ e_{\tau_i+1}>i\geq e_{\tau_i}\geq\tau_i,\ \tau_{e_{\tau_i}}=\tau_i\\ \forall i,\ 1\leq i\leq n-1,\ \tau_i\leq\tau_{i+1}\leq\tau_i+1\end{cases} \tag{28}$$

Par convention, on pose: $e_0=0$.

On a:

$$\forall i,\quad i\in E' \Rightarrow d'_i>c'_i$$

$$\Rightarrow d_i+\min(i,\lambda)D-\sum_{k=1}^{\sigma_i}\Delta_k>[\min(i,\lambda)+i]D$$

$$\Rightarrow d_i>iD+\sum_{k=1}^{\sigma_i}\Delta_k=c_i$$

En particulier:

$$\forall i,\quad e_{\tau_i}\in E' \Rightarrow d_{e_{\tau_i}}>c_{e_{\tau_i}} \tag{29}$$

On déduit de (28), (29) et (21):

$$\forall i,\ d_i\geq c_{e_{\tau_i}}=e_{\tau_i}D+\sum_{k=1}^{\sigma_{e_{\tau_i}}}\Delta_k$$

formule que l'on utilise pour minorer (24):

$$\forall i,\quad d'_i=d_i+\min(i,\lambda)D-\sum_{k=1}^{\sigma_i}\Delta_k$$

$$\geq e_{\tau_i}D+\sum_{k=1}^{\sigma_{e_{\tau_i}}}\Delta_k+\min(i,\lambda)D-\sum_{k=1}^{\sigma_i}\Delta_k$$

$$=e_{\tau_i}D+\min(i,\lambda)D-\sum_{k=\sigma_{e_{\tau_i}}+1}^{\sigma_i}\Delta_k \tag{30}$$

Par ailleurs (23) et (24) impliquent:

$$d'_i=d_i+\min(i,\lambda)D-\sum_{k=1}^{\sigma_i}\Delta_k$$

$$\geq h_{\sigma_i}D+\sum_{k=1}^{\sigma_i}\Delta_k+\min(i,\lambda)D-\sum_{k=1}^{\sigma_i}\Delta_k$$

$$=[h_{\sigma_i}+\min(i,\lambda)]D \tag{31}$$

Avec (30) et (31) on a donc:

$$d'_i \geq \max(h_{\sigma_i}D, e_{\tau_i}D - \sum_{k=\sigma_{e_{\tau_i}}+1}^{\sigma_i} \Delta_k) + \min(i,\lambda)D$$

Nous cherchons maintenant à simplifier cette minoration en distinguant deux cas:

$$1)\ \sigma_{e_{\tau_i}} = \sigma_i \Rightarrow e_{\tau_i} \geq h_{\sigma_i} \text{ et } \sum_{k=\sigma_{e_{\tau_i}}+1}^{\sigma_i} \Delta_k = 0$$

$$\Rightarrow d'_i \geq [e_{\tau_i} + \min(i,\lambda)]D$$

$$2)\ \sigma_{e_{\tau_i}} < \sigma_i \Rightarrow e_{\tau_i} < h_{\sigma_i} \text{ et } \sum_{k=\sigma_{e_{\tau_i}}+1}^{\sigma_i} \Delta_k > 0$$

$$\Rightarrow e_{\tau_i}D - \sum_{k=\sigma_{e_{\tau_i}}+1}^{\sigma_i} \Delta_k < h_{\sigma_i}D$$

$$\Rightarrow d'_i \geq [h_{\sigma_i} + \min(i,\lambda)]D$$

Dans tous les cas, on a:

$$\forall i,\ d'_i \geq [\max(h_{\sigma_i}, e_{\tau_i}) + \min(i,\lambda)]D \qquad (32)$$

On construit alors un problème P" à partir du problème P' en conservant les dates d'arrivée et les durées des tâches et en modifiant les délais de la manière suivante:

$$\forall i,\ i \in E',\quad d''_i = +\infty > d'_i$$

et $$\forall i,\ i \notin E',\quad d''_i = [\max(h_{\sigma_i}, e_{\tau_i}) + \min(i,\lambda)]D \quad (\leq d'_i \text{ d'après (32)})$$

Alors en utilisant le corollaire 2 et (27), on obtient:

$$T - T^* \leq T(P',O') - T(P',O^*_{P'}) \leq T(P'',O') - T(P'',O^*_{P''}) \qquad (33)$$

Pour P", on peut facilement construire un ordonnancement minimal O''^m (non réalisable, voir Fig.3.3) en plaçant les tâches n'appartenant pas à E' dans l'ordre de leur numérotation, les unes derrière les autres, sans laisser la machine inoccupée, et en plaçant les tâches de E' immédiatement derrière dans n'importe quel ordre puisqu'elles ne sont jamais en retard.

En fait, cet ordonnancement O''^m vérifie l'ordre EDD (Earliest Due Date), parce que:

$$\forall i,j,\ i \notin E',\ j \notin E',\ j > i:$$

$$\min(j,\lambda) \geq \min(i,\lambda) \qquad (34)$$

et (22) et (28) $$\Rightarrow e_{\tau_j} \geq e_{\tau_i} \text{ et } h_{\sigma_j} \geq h_{\sigma_i}$$

$$\Rightarrow \max(h_{\sigma_j}, e_{\tau_j}) \geq \max(h_{\sigma_i}, e_{\tau_i}) \qquad (35)$$

Avec (34) et (35), on a:

$\forall i,j,\ i \notin E',\ j \notin E',\ j>i,$

$\max(h_{\sigma_j},e_{\tau_j})+\min(j,\lambda) \geq \max(h_{\sigma_i},e_{\tau_i})+\min(i,\lambda) \Rightarrow d''_j \geq d''_i$

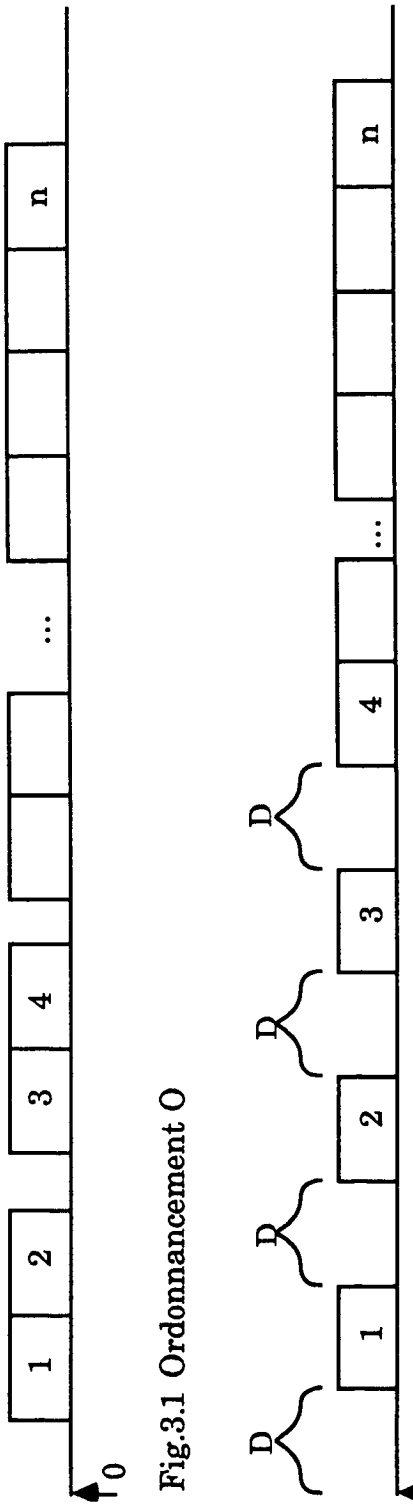

Fig.3.1 Ordonnancement O

Fig.3.2 Ordonnancement O'

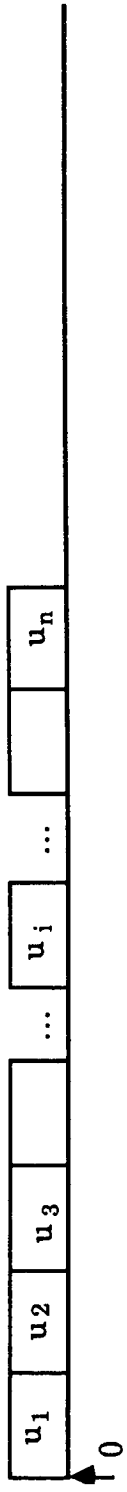

Fig.3.3 Ordonnancement O''^m avec $d_{u_1} \leq d_{u_2} \leq \ldots \leq d_{u_n}$

Cet ordonnancement a une somme des retards inférieure ou égale à celle de l'ordonnancement optimal du problème P", car on sait que EDD est optimal pour le problème du type $n/1/p_i=1,r_i=0/\Sigma T_i$ ce qui permet de déduire:

$$T(P",O"^m)\leq T(P",O^*_{P"}) \Rightarrow T(P",O')-T(P",O^*_{P"})\leq T(P",O')-T(P",O"^m) \quad (36)$$

Pour finir on majore $T(P",O')-T(P",O"^m)$:

$\forall i,\ c'_i=[i+\min(\lambda,i)]D$

$$T_i(P",O')=\begin{cases}0 \text{ si } i\in E'\\ c'_i-d"_i=[i-\max(h_{\sigma_i},e_{\tau_i})]D \text{ si } i\notin E'\end{cases}$$

et $\forall i,\ i\notin E'\ c"^m_i=[i-\tau_i]D$

On a donc:

$$\forall i,\ T_i(P",O"^m)=\begin{cases}0 \text{ si } i\in E'\\ \max(c"^m_i-d"_i,0)=\max[i-\tau_i-\min(\lambda,i)-\max(h_{\sigma_i},e_{\tau_i}),0]D \text{ si } i\notin E'\end{cases}$$

on a:

$$\begin{aligned}\forall i,\ i\notin E',\ T_i(P",O')-T_i(P",O"^m) &=\{i-\max(h_{\sigma_i},e_{\tau_i})-\max[i-\tau_i-\min(i,\lambda)-\max(h_{\sigma_i},e_{\tau_i}),0]\}D\\ &=-\max[-\tau_i-\min(i,\lambda),\max(h_{\sigma_i},e_{\tau_i})-i]D\\ &=\min[\tau_i+\min(i,\lambda),i-\max(h_{\sigma_i},e_{\tau_i})]D\\ &=\min[\tau_i+\min(i,\lambda),-\max(h_{\sigma_i}-i,e_{\tau_i}-i)]D\\ &=\min[\tau_i+\min(i,\lambda),\min(i-h_{\sigma_i},i-e_{\tau_i})]D\\ &=\min[\min(i+\tau_i,\lambda+\tau_i),\min(i-h_{\sigma_i},i-e_{\tau_i})]D\\ &=\min(i+\tau_i,\lambda+\tau_i,i-h_{\sigma_i},i-e_{\tau_i})D\\ &=\min(\lambda+\tau_i,i-h_{\sigma_i},i-e_{\tau_i})D \ (1) \quad (37)\end{aligned}$$

et $\forall i,\ i\in E',\ T_i(P",O')-T_i(P",O"^m)=0=\min(\lambda+\tau_i,i-h_{\sigma_i},i-e_{\tau_i})D$ (2) (38)

De (37) et (38) on déduit:

$$\begin{aligned}T(P",O')-T(P",O"^m)&=\sum_{i=1}^{n}[T_i(P",O')-T_i(P",O"^m)]\\ &=\sum_{i=1}^{n}\min(\lambda+\tau_i,i-h_{\sigma_i},i-e_{\tau_i})D\end{aligned}$$

Il reste donc à chercher la borne supérieure de $\sum_{i=1}^{n}\min(\lambda+\tau_i,i-h_{\sigma_i},i-e_{\tau_i})D$.

---

(1) En fait, on a toujours $\tau_i\geq 0$, $i-h_{\sigma_i}\leq i$ et $i-e_{\tau_i}\leq i$, on a donc: $\min(i-h_{\sigma_i},i-e_{\tau_i})\leq i+\tau_i$

(2) Pour tout $i\in E'$, $e_{\tau_i}=i$, $\lambda+\tau_i\geq\lambda>0$ et $i\geq h_{\sigma_i}$, on a donc: $\min(\lambda+\tau_i,i-h_{\sigma_i},i-e_{\tau_i})D=0$

Quatrième Partie:

On doit résoudre le problème de maximisation suivant:

$$\text{Max}\sum_{i=1}^{n}\min(\lambda+\tau_i, i-h_{\sigma_i}, i-e_{\tau_i})$$

sous les contraintes:

$$\begin{cases} h_{\sigma_1}=\sigma_1=1,\ \sigma_n=m\geq\lambda,\ \tau_n=\omega \\ \forall i,\ \sigma_{h_{\sigma_i}}=\sigma_i,\ \tau_{e_{\tau_i}}=\tau_i \\ \forall i,\ h_{\sigma_i+1}>i\geq h_{\sigma_i}\geq\sigma_i,\ e_{\tau_i+1}>i\geq e_{\tau_i}\geq\tau_i \\ \forall i,\ 1\leq i\leq n-1,\ \sigma_i\leq\sigma_{i+1}\leq\sigma_i+1,\ \tau_i\leq\tau_{i+1}\leq\tau_i+1 \end{cases}$$

Ce problème a une solution unique qui est:

$$\begin{cases} \forall i,\ i\leq\lambda,\ \sigma_i=h_{\sigma_i}=i \\ \forall i,\ i>\lambda,\ \sigma_i=h_{\sigma_i}=\lambda \\ \forall i,\ i\leq\omega,\ \tau_i=e_{\tau_i}=i \\ \forall i,\ i>\omega,\ \tau_i=e_{\tau_i}=\omega \end{cases}$$

Avec (33) et (36) on a donc:

$$\begin{aligned} T-T^* &\leq T(P',O')-T(P',O^*_{P'}) \\ &\leq T(P'',O')-T(P'',O^*_{P''}) \\ &\leq T(P'',O')-T(P'',O''^m) \\ &=\sum_{i=1}^{n}[T_i(P'',O')-T_i(P'',O''^m)] \\ &=\sum_{i=1}^{n}\min(\lambda+\tau_i, i-h_{\sigma_i}, i-e_{\tau_i})D \\ &\leq \sum_{i=1}^{\max(\lambda,\omega)}0 + \sum_{i=\max(\lambda,\omega)+1}^{n}\min(\lambda+\omega, i-\lambda, i-\omega)D \\ &= \sum_{i=\max(\lambda,\omega)+1}^{n}\min(\lambda+\omega, i-\lambda, i-\omega)D \\ &= \sum_{i=\max(\lambda,\omega)+1}^{n}\min[\lambda+\omega, i-\max(\lambda,\omega)]D \\ &= \sum_{i=1}^{n-\max(\lambda,\omega)}\min(\lambda+\omega, i)D \end{aligned}$$

$$= \sum_{i=1}^{n-\max(\lambda,\omega)} \min[\max(\lambda,\omega)+\min(\lambda,\omega),i]D$$

Pour faciliter le calcul, on pose:

$n1=\min(\lambda,\omega)$ et $n2=\max(\lambda,\omega)$

Alors on a:

$1 \le n1 \le n2 \le n$

et $$T-T^* \le \sum_{i=1}^{n-n2} \min(n1+n2,i)D$$

Et on doit chercher la valeur maximale de la fonction

$$f(n1,n2)= \sum_{i=1}^{n-n2} \min(n1+n2,i)$$

en examinant les deux cas suivants.

A) $n-n2 \le n1+n2$

Alors:

$$f(n1,n2)= \sum_{i=1}^{n-n2} i = \frac{(n-n2+1).(n-n2)}{2}$$

On a en outre: $n-2n2 \le n1 \le n2 \Rightarrow n2 \ge \frac{n}{3}$,

d'où $$f(n1,n2) \le \frac{(n-\frac{n}{3}+1).(n-\frac{n}{3})}{2}$$

$$= \frac{2n^2+3n}{9}$$

$$\le \frac{(2n+1)^2}{16}$$

B) $n-n2 > n1+n2$

Alors:

$$f(n1,n2)= \sum_{i=1}^{n1+n2} i + \sum_{i=n1+n2+1}^{n-n2} (n1+n2)$$

$$= \frac{(n1+n2).(n1+n2+1)}{2} + [n-(n1+2n2)].(n1+n2)$$

$$= \frac{(n1+n2).(2n+1-n1-3n2)}{2}$$

$$= \frac{(4n1+4n2).[2(2n+1)-2n1-6n2]}{16}$$

$$= \frac{[(2n+1+n1-n2)-(2n+1-3n1-5n2)].[(2n+1+n1-n2)+(2n+1-3n1-5n2)]}{16}$$

$$= \frac{[2n+1+n1-n2]^2-[2n+1-3n1-5n2]^2}{16} \le \frac{[2n+1+n1-n2]^2}{16}$$

$$\le \frac{(2n+1)^2}{16}$$

Dans tous les cas, on a:

$$T\text{-}T^* \leq f(n1,n2)D \leq \frac{(2n+1)^2}{16}D$$

Q.E.D.

5. Conclusion

Nous avons défini un nouvel ensemble d'ordonnancements et prouvé qu'il contient au moins une solution optimale, ce qui nous permet de travailler sur un sous-ensemble plus restreint.

Au lieu de l'explorer (au moins implicitement) nous avons proposé de construire, de manière polynômiale de "bons" ordonnancements de cet ensemble.

Un majorant de l'erreur commise dans le pire des cas a été calculé dans un cas particulier.

Les résultats ont été obtenus pour le problème d'ordonnancement à une machine $n/1/r_i/\sum T_i$. Il nous paraît intéressant de chercher à étendre les notions définies ici au cas général à plusieurs machines ($n/m/r_i/\sum T_i$). Dans ce cas, il est nécessaire d'introduire des délais par opération, et comme ces délais sont plus ou moins arbitraires, nous perdons la propriété de dominance des ensembles d'ordonnancements construits. Nous effectuons actuellement nos recherches dans ce sens et cela nous incite à chercher à intégrer un indicateur d'urgence généralisé comme règle de choix dans un système expert.

6. Annexe: étude analytique de la fonction PRIOR

Par définition, la fonction PRIOR est donnée par la formule:

$$PRIOR(i,\Delta)=\max(\Delta,r_i) + \max[\max(\Delta,r_i)+p_i, d_i]$$

En étudiant les différents cas possibles et en supposant que l'on travaille sur les délais corrigés ($d_i \geq r_i+p_i$), on obtient immédiatement que:

$$PRIOR(i,\Delta)=\begin{cases} r_i+d_i \text{ si } \Delta \leq r_i \\ \Delta+d_i \text{ si } r_i \leq \Delta \leq d_i-p_i \\ 2\Delta+p_i \text{ si } d_i-p_i \leq \Delta \end{cases}$$

C'est une fonction non décroissante, continue et linéaire par morceaux (voir Fig. 4.1). Elle est constante tant que la tâche i n'est pas arrivée. Elle croît à la même vitesse que Δ tant que la tâche planifiée au plus tôt n'est pas en retard et son ordonnée à l'origine est d_i. Aussi, si on compare sa priorité avec celle d'une autre tâche disponible et non en retard en commençant à l'instant Δ, c'est la tâche de plus petit délai corrigé qui est la plus prioritaire. Elle croît à une vitesse double de Δ lorsque la tâche planifiée au plus tôt est en retard et son ordonnée à l'origine est p_i. Aussi, si on compare sa priorité avec celle d'une autre tâche qui serait aussi en retard en commençant en Δ, c'est la tâche de plus petite durée qui est la plus prioritaire.

On peut considérer que cette règle de priorité est une règle souple qui lorsque l'on compare deux tâches à placer consécutivement sur une machine, passe de la règle FIFO (First In First Out) quand une des deux tâches n'est pas encore sur le point d'arriver, à la règle EDD (Earliest Due Date) quand aucune des deux tâches n'est en retard, puis à la règle SPT (Shortest Processing Time) lorsque les deux tâches sont en retard. Les cas transitoires (par exemple, l'une en retard et pas l'autre) sont traités également puisque $PRIOR(i,\Delta) \leq PRIOR(j,\Delta) \Rightarrow T_{i,j}(\Delta) \leq T_{j,i}(\Delta)$. Les instants où la priorité change peuvent être obtenus analytiquement dans chacun des cas possibles comme le montrent les figures 4.2, 4.3 et 4.4.

Toutes les figures ont été réalisées en supposant arbitrairement que la tâche d'indice i est la tâche ayant le plus petit délai ($d_i \leq d_j$).

La figure 4.2. correspond au cas où $r_i+d_i \leq r_j+d_j$ (ou encore $d_j-d_i \geq r_i-r_j$, c'est-à-dire que l'écart entre les délais est supérieur à l'écart entre les dates d'arrivées en inversant les tâches et quelle que soit la tâche qui arrive la première). Dans ce cas, la tâche de plus petit délai est toujours prioritaire si c'est également la tâche de plus petite durée ($p_i \leq p_j$) ; sinon la tâche de plus petit délai est la plus prioritaire jusqu'à l'instant d_j-p_i (à partir duquel les deux tâches seront en retard), puis celle de plus petite durée devient la plus prioritaire.

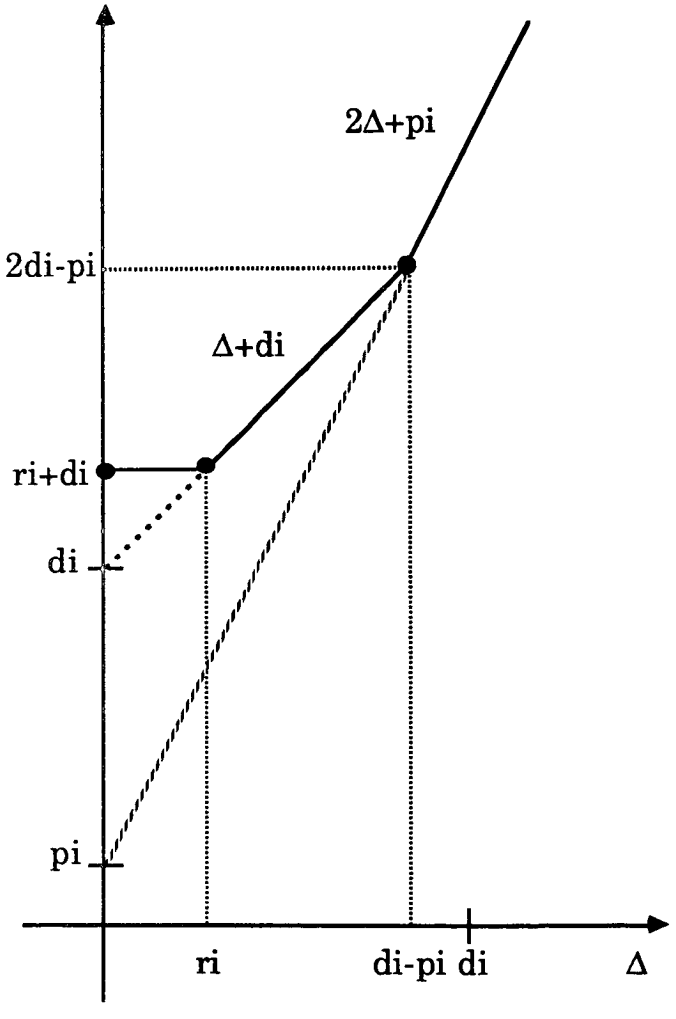

Fig. 4.1.

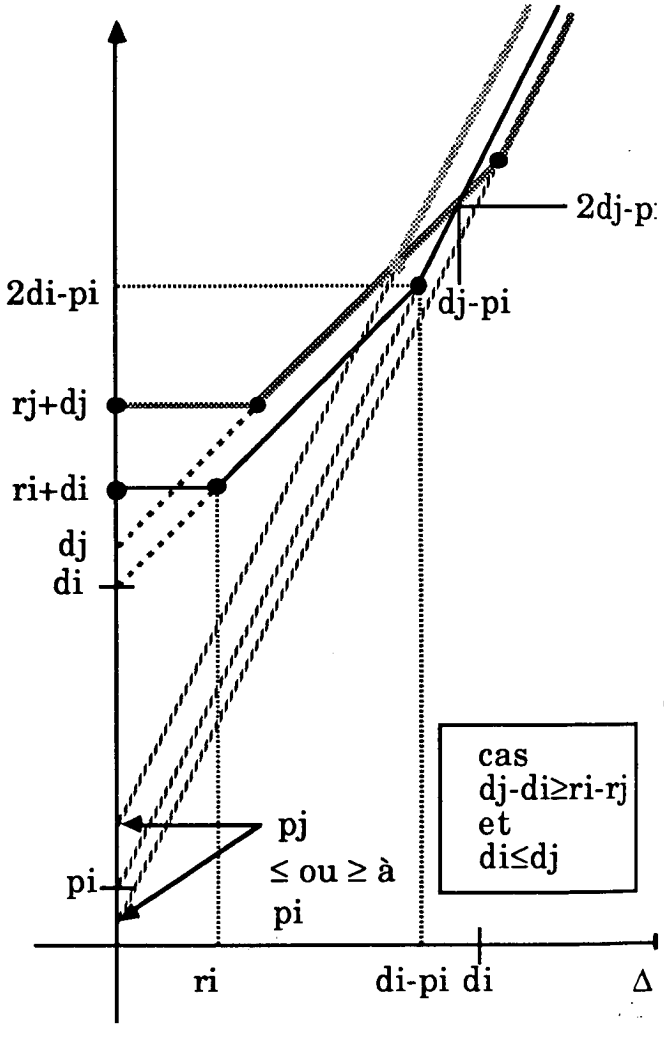

Fig. 4.2.

Les figures 4.3. et 4.4. correspondent au cas où $r_i+d_i \geq r_j+d_j$ (ou encore $d_j-d_i \leq r_i-r_j$, c'est-à-dire que l'écart entre les délais est inférieur à l'écart entre les dates d'arrivées en inversant les tâches). Dans ces deux cas, c'est obligatoirement la tâche de plus grand délai qui arrive la première.

La figure 4.3. correspond au sous cas où $r_i+d_i \leq 2d_j-p_j$. Alors, la tâche qui arrive le plus tôt est la plus prioritaire jusqu'à l'instant $r_i+d_i-d_j$ (c'est-à-dire la date d'arrivée de celle de plus petit délai diminuée de l'écart entre les délais). Puis la tâche de plus petit délai devient la plus prioritaire et le reste si c'est aussi la tâche de plus courte durée. Sinon, c'est l'autre tâche de plus courte durée qui redevient prioritaire à partir de l'instant d_j-p_i comme dans le cas de la figure 4.2.

La figure 4.4. correspond au sous cas où $r_i+d_i \geq 2d_j-p_j$. Alors, la tâche qui arrive le plus tôt est la plus prioritaire jusqu'à l'instant $\frac{r_i+d_i-p_j}{2}$. Puis c'est la tâche de plus petit délai qui a également dans ce cas la plus petite durée qui devient la plus prioritaire et qui le reste.

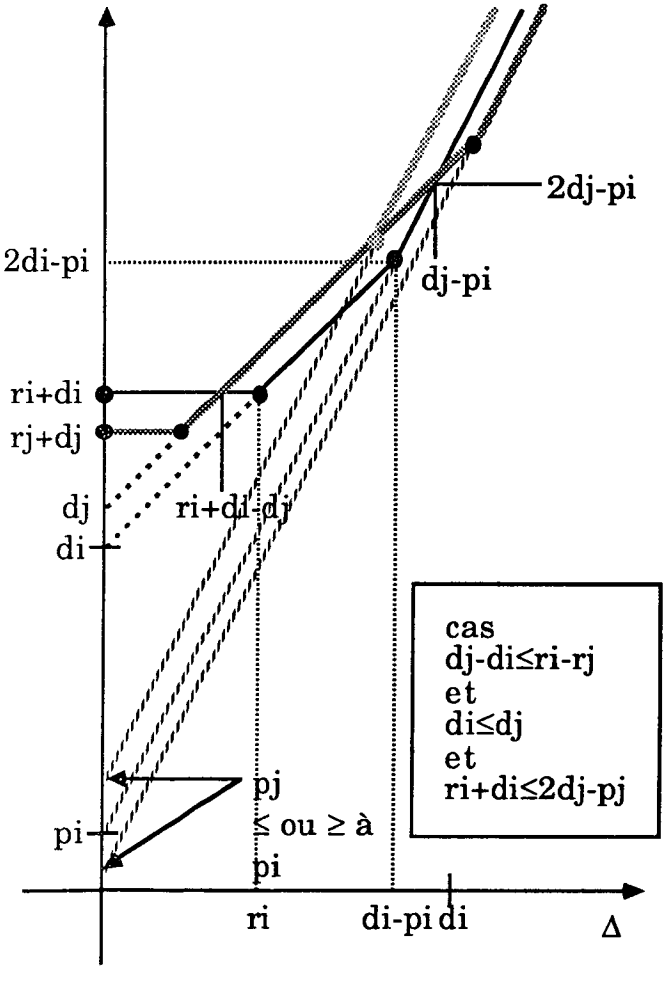

Fig. 4.3.

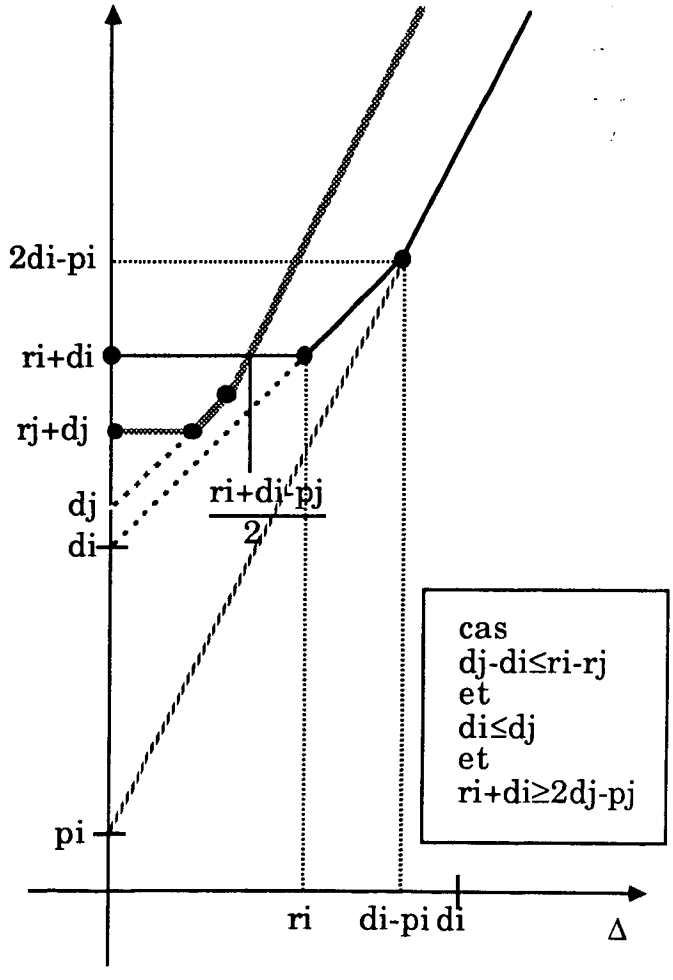

Fig. 4.4.

Cette étude analytique montre bien que si l'on s'intéresse à l'évolution de la priorité de deux tâches consécutives au cours du temps avec pour objectif de minimiser la somme des retards alors la fonction PRIOR donne d'abord la priorité à la tâche qui arrive la première (FIFO), puis donne la priorité à celle de plus petit délai (EDD) et enfin, lorsque les deux tâches sont en retard, elle donne la priorité à celle de plus petite durée (SPT).

REFERENCES

[1] **K.R. BAKER**
Introduction to sequencing and scheduling
John & Wiley, 1974

[2] **R.W. CONWAY / W.C. MAXWELL / L.W. MILLER**
Theory of scheduling
Addison-Wesley, 1967

[3] **J.R. JACKSON**
"Scheduling a production line to minimize maximum tardiness"
Research Report 43, Management Science, Research Report,
University of California, Los angeles, 1955

[4] **A.H.G. RINNOOY KAN**
"Machine sequencing problems: Classification, complexity and computation"
Nijhoff, The Hague, 1976

[5] **W.E. SMITH**
"Various optimizers for single-stage production"
Nav. Res. Log. Quart. Vol. 3, pp 59-66, 1956

[6] **V. SRINIVASAN**
"A hybrid algorithm for the one-machine sequencing problem to minimize total tardiness"
Nav. Res. Log. Quart. Vol. 18, n° 3 pp 317-327, 1971

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique